Refactoring CA Gen applications:

What, When, Why, Where and How

Ryan Johnson

EDGEucate 2007

Session 2, Monday 29th Oct 2007, 10:15

delivering strategic enterprise solutions

Facet profile

CA Gen specialists based in Brisbane, Australia

Clients across Europe, Australia and SE Asia

Recently acquired Response Systems

Services include

  Upgrades to CA Gen infrastructure and applications

  Offsite development from our Brisbane labs

  Consultancy on process, architecture and SDLC

  Strategic reviews

delivering strategic enterprise solutions

Session objectives

Understand what Refactoring is

Understand why it can be valuable – and when

Learn where others have started, and importantly…

What the foundations of success for this are!

Refactoring case study

Undertaken over the last three years

Collaborative approach

  Partnership - open and transparent

  Incremental – initial small steps

  Jointly laying the foundations for success

  Reviewing and evaluating as we progressed

This session is drawn from the real-world

What is Refactoring?

Refactoring is the <u>process</u> of restructuring an application from the inside-out, without functionally altering the application.

Business change is not introduced

Refactored applications require regression testing

This ensures the risk profile of this activity is low

delivering strategic enterprise solutions

When to Refactor?

So what's the justification?

There's no direct benefit to the business

It's essentially a "technical" exercise!

delivering strategic enterprise solutions

When to Refactor?

So what's the justification?

- There's no direct benefit to the business
- It's essentially a "technical" exercise!

There are a number of different indicators:

- Old architectures constraining new directions
- Inconsistent or absent standards (or enforcement)
- Increasing cost to deliver the same volume of change
- Increasing cost of (impact) analysis and test effort

When to Refactor?

For our client it was a combination of these:

- Volume of business change was reasonably constant
- Cost to implement that change was continually rising
- Entire application was contained within a single model
- 3270 screens delivered from psteps that accessed data
- Testing impact on small change was large (or unknown)
- Application architecture was preventing much reuse
- Many different ways of resolving the same problems

Why Refactor?

Accepting that "something must change"

Usually driven by cost considerations

Understand the alternatives:

Replace	Risk moderate; Cost high
Rewrite	Risk very high; Cost high – very high
Refactor	Risk low - moderate; Cost moderate

Refactoring is the lowest cost and risk of the three

Why Refactor?

Refactoring is the lowest cost and risk - but only if

- The objectives from Refactoring are defined
- The benefits and outcomes are defined and measurable
- The project costs and risks are contained

And these are the factors upon which a strong business case can be built, approved and supported:

Refactoring requires an investment but will yield a greater return than the investment made

Where to start?

1. Define where you're going

2. Define the process to get you there

3. Choose a functional area to commence with

delivering strategic enterprise solutions

Define where you're going

Make the big decisions first

      Consider moving to components

      Define the component architecture

      Implementation of Services Oriented Architecture

      Functionally "tier"ing the application

      Reduce coupling but retain cohesion

      Keep it "real" and avoid too much architectural "purity"

delivering strategic enterprise solutions

Define where you're going

Define the target outcomes

- New or revised architecture definitions
- New or revised coding standards that support those
- Coding templates that implement the new standards
- Produce a worked example as a real reference

Determine how to objectively measure the outcomes

Communicate with the existing developer community

delivering strategic enterprise solutions

Define the process

It's a programme of work so the activity must be

- Defined and repeatable
- Scalable so that you can "divide and conquer"
- Incorporate how ongoing work meshes with Refactoring
- Define the model management strategies
- Define the testing and acceptance and deployment

delivering strategic enterprise solutions

Choose a starting point

Execute the process against a single functional area

- Not so small or simple that the difficulties are not faced
- Not so large that the first iteration isn't achievable
- Ideally a 3-6 month effort that deploys into Production

delivering strategic enterprise solutions

Choose a starting point

Execute the process against a single functional area

- Not so small or simple that the difficulties are not faced
- Not so large that the first iteration isn't achievable
- Ideally a 3-6 month effort that deploys into Production

Review and revise

- Collect the metrics and analyse what worked and did not
- Modify the process if required
- Fill in the gaps in the architecture and standards

delivering strategic enterprise solutions

So how did this work?

Started by defining a Refactoring process

Identified some key enablers prior to commencing

- Define the tiers to implement within the application
- Defined a component hierarchy
- Expanded coding standards to cover CBD
- Revised standards to support these with checklists
- Acceptance testing strategies
- Metrics collection for measuring improvements

So how did this work?

Chose a complete functional area

- Indicative complexity
- Clear business value

Split it into three prioritised clusters

- Update/Create/Delete functions around "core entities"
- The remaining Update/Create/Delete functionality
- The more widely utilised Read integration points

Each executed as a project with standard controls

What were the lessons?

The application functionality should change!

    Existing behaviour was inconsistent

    Error messages, Screen layouts, Function keys

The goal of a more consistent application was at odds with avoiding functional change

Testing without automation pushed costs up

What changed?

Allowed change – but controlled it through design

    The business would approve the change

    The testers then knew to expect the change

    Consistency then supported Refactoring objectives!

Invested heavily in automated regression testing

Plugged gaps in the Architecture and Standards

The next iteration used the revised approach

The road so far:

First functional area now Refactored

> Fully componentised

> Functional area split into presentation, rules and db tiers

Automated testing tools now in place

Process improvements also in use by core teams

Many related activities improved for everyone:

> Change control and model management

> Definition of test cases

delivering strategic enterprise solutions

Measured by

External function point analysis performed

> Committed to ongoing measurement - quarterly

> Commenced with a baseline measurement

> Reviewed after each major quarterly release

Shows empirically that cost / function point change is decreasing in the areas that have been refactored

Business has committed to ongoing investment

delivering strategic enterprise solutions

Summary

Refactoring can extend the life of large applications

      Reducing ongoing cost of ownership

      Simplifying application structure

      Enabling faster business change – time to market

      Enabling exploitation of new technologies and platforms

Significantly lower risk and cost than replacement

Determine <u>how</u> you're going to measure progress

Define your goals, then the process and then begin!

delivering strategic enterprise solutions

Questions?

Email: <u>ryan.johnson@facet.com.au</u>

delivering strategic enterprise solutions